CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

Deductive Synthesis of Concurrent Construction Plans

by

Zohar Manna,
Massimo Paltrinieri,
Richard Waldinger

TECHNICAL REPORT
Number 75

September, 1992

Stanford University

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 94305-4020

SUMMARY
CIFE TECHNICAL REPORT # 75

Title:	Deductive Synthesis of Concurrent Construction Plans
Authors:	Zohar Manna, Massimo Paltrinieri, Richard Waldinger
Publication Date:	September 1992
Funding Sources:	NSF, under grants CCR-89-11512 and CCR-89-13641
	DARPA, under contract NAG2-703
	CIFE, partial seed research grant

1. Abstract: In the deductive approach, the synthesis of a plan is regarded as a problem in theorem proving. The goal state is described by a sentence in first-order logic and a correct plan is extracted from a proof of the sentence. A deductive framework, obtained by adapting a situational calculus for automated planning, is applied to the formation of construction plans for civil engineering. Proofs of theorems for even simple construction problems have required an expressive language to represent the structure of plans. Desirable features include sequencing, parallelism, contingency, repetition and modularity.

2. Subject: This research investigates the use of first-order logic to construction-planning problems. The choice of this framework is motivated by the high expressive power required by these problems. For instance, when a condition on the construction process, such as resource availability, is not known at planning time, the plan must account for contingency; when the same pattern is contained several times in a structure, such as a multi-story building, the plan must account for repetition.

3. Objectives/Benefits: Current construction-planning systems have limitations in their expressive power that lead to the impossibility of including desirable features in the derived plan. This report describes a framework that avoids these limitations and presents examples on the use of this framework in the construction domain.

4. Methodology: The construction plan illustrated in this report was derived with the Deductive Tableau System. Originally, the system was an interactive implementation of a theorem prover for first-order logic. Then it was extended to allow the synthesis of programs. Presently, we are testing a new version of the system that generates plans either automatically or interactively. In the latter case, the user can control the derivation process to introduce desired features into the plan.

5. Results:

- First-order logic is a suitable formalism for construction planning
- It allows the introduction of sequencing, parallelism, contingency, repetition and modularity in the construction plans
- Much of the derivation process can be automated
- For this purpose, the Deductive Tableau System is a valuable support

6. Research Status: Deductive frameworks based on first-order logic seem appropriate for the derivation of construction plans. Nevertheless, before the process can reach a high level of automation, effective domain-dependent strategies must be developed to explore the potentially huge search space produced by real-life applications. Also, to reduce the fragmentation between the design and planning phases of the project, the planning system should be integrated with CAD software to automatically extract the axioms describing the structure of the facility to be constructed from purely geometrical information.

Deductive Synthesis of Concurrent Construction Plans *

Zohar Manna
Department of Computer Science
Stanford University
Stanford, CA 94305
zm@cs.stanford.edu

Massimo Paltrinieri
Department of Computer Science
Stanford University
Stanford, CA 94305
palmas@cs.stanford.edu

Richard Waldinger
Artificial Intelligence Center
SRI International
Menlo Park, CA 94025
waldinger@ai.sri.com

Abstract

In the deductive approach, the synthesis of a plan is regarded as a problem in theorem proving. The goal state is described by a sentence in first-order logic and a correct plan is extracted from a proof of the sentence.

A deductive framework, obtained by adapting a situational calculus for automated planning, is applied to the formation of construction plans for civil engineering. Proofs of theorems for even simple construction problems have required an expressive language to represent the structure of plans. Desirable features include sequencing, parallelism, contingency, repetition and modularity.

1 Introduction

Plan synthesis [FN71, Gre69, Ros81, Wal81] concerns the generation of some course of actions to achieve a specified goal. A domain-independent framework for the deductive synthesis of plans is proposed in [MW87b].

In this approach, the generation of a plan is regarded as a task in theorem proving; that is, the plan is extracted from the proof of a theorem that extablishes the existence of a final state in which the goal condition is true. The proof is restricted to be sufficiently constructive, in that it must describe a computational method for finding the final state.

*This research was supported in part by the National Science Foundation under grants CCR-89-11512 and CCR-89-13641, by the Defense Advanced Research Projects Agency under contract NAG2-703 and by a partial seed research grant from the Center for Integrated Facility Engineering, Stanford University.

This method becomes the basis for the plan extracted from the proof. The structure of the proof determines the structure of the plan, which is guaranteed to be correct by the soundness of the deduction rules.

Research on applying symbolic approaches to construction-planning problems has been conducted only recently (for a comprehensive review see [Kar89]). The system CONSTRUCTION PLANEX [HZGR+87] generates project networks defining precedence among and duration of activities. GHOST [NSL88] reasons about objects in the construction domain to define project activities and precedence relations. OARPLAN [DLHR89] takes as its input a description of the facility to be constructed and generates a hierarchical project plan for construction of the facility. The use of SIPE-2 to generate hierarchical plans for the construction of a single-family house is discussed in [KLW91].

As pointed out in [Geo87], a plan usually has a definite structure that depends on how it has been composed from more primitive components. The standard ways of composing plans include *sequencing* (resulting in sequential plans), *parallelism* (concurrent plans), *contingency* (conditional plans), *repetition* (repetitive plans) and *modularity* (hierarchical plans). Most of the planning systems to date have been oriented towards the generation of plans which are fully or partially ordered sequences of primitive actions (for a review of AI planning techniques see [THD90]). Less research has concentrated on the generation of plans which contain contingency and repetition, and the problem has not been addressed in the construction domain yet.

We believe that this is an important limitation, for instance, when a condition on the construction process, such as resource availability, is not known at planning time (contingency) or when the same pattern is contained several times in a structure, such as a multi-story building (repetition). Our logical framework allows for all of the mentioned features to be introduced into synthesized plans from structural and topological descriptions of the facility to be constructed.

In the next section, the logical framework developed [MW87b, MW87a] for the deductive synthesis of plans is outlined. A theory for the construction domain is developed within this framework in Section 3. The introduction of relevant features in construction plans is illustrated in Sections 4–8. A complete proof, from which a high-level construction plan for a multi-story building is extracted, is finally presented in the Appendix. The plan employs all the mentioned features. The proof is split into two parts: the first part generates the top-level plan *construct*(b) to construct a multi-story building b; the second part extracts a more detailed subplan *constr*(*story*) to construct a single story of the building.

2 Derivation Process

In our deductive approach, the construction of a plan is regarded as the proof of a theorem in a variant of situational calculus [MH69]. In this framework, there are three classes of terms: *state terms*, *object terms* and *plan terms*. Plan terms are also called *fluent terms*. A fluent may be thought of as a function ranging over states: evaluating a fluent in a given state *returns* an object and *produces* a new state. If s is a state term and e is a fluent term, then $s{:}e$ is the object returned by evaluating e in state s and $s;e$ is the state produced by evaluating e in state s.

Example (term evaluation). Given a state term s_0, a fluent term b denoting a building, and the fluent terms *constructed*(b) and *construct*(b), with the intuitive meaning, we have that

- $s_0{:}b$ is the object returned by evaluating b in state s_0, that is, the building denoted by b
- $s_0{:}$*constructed*(b) is the truth-value returned by evaluating *constructed*(b) in s_0, depending on whether b is constructed or not in state s_0
- $s_0;$*construct*(b) is the state produced by evaluating *construct*(b) in s_0, that is, the state that results from the construction of b. □

If e is such that $s;e{=}s$ for any state s, i.e., the evaluation of e in a given state never produces a new state, then e is said to be an *applicative designator*. On the other hand, if $s{:}e{=}e$ for any state s, i.e., the evaluation of e returns the same value in every state, then e is said to be a *rigid designator*.

Example (applicative and rigid designators). Consider the fluent terms introduced in the previous example.

- b is applicative, since its evaluation does not change the state
- *construct*(b) is not applicative, since its evaluation may change the state
- b is also a rigid designator, since its value is the same in any state
- *constructed*(b) is not rigid, since its value may change from state to state. □

For every plan term $f(u_1,..,\ u_n)$, where $u_1,..,\ u_n$ are fluent terms, we shall introduce a corresponding state term $;f(x_1,..,\ x_n,\ s)$, where $u_1,..,u_n$ and $x_1,..,x_n$ are object terms, while s is a state term. Their correspondence is given by the following *plan linkage* axiom schema

$$s;f(u_1,..,\ u_n) = ;f(s{:}u_1,..,\ s{:}u_n,\ s)$$

Example (plan linkage axiom). The plan term $place(u)$, where u is applicative, has a corresponding object term $;place(x,s)$. Such a correspondence is determined by the *place linkage* axiom

$$s;place(u) = ;place(s{:}u,\ s)$$

an appropriate instance of the *plan linkage* axiom schema. Also the plan term $construct(u)$, where u is applicative, has a corresponding object term $;construct(x,s)$. The appropiate instance of the *plan linkage* axiom schema this time is

$$s;construct(u) = ;construct(s{:}u,\ s)$$

referred to as the *construct linkage* axiom. □

To derive a plan for achieving a condition $\mathcal{Q}[s_0,\ a,\ s_f]$, where s_0 is the initial state, a the input object, and s_f the final state, we prove the theorem

$$(\forall a)(\exists z_1)(\forall s_0)\mathcal{Q}[s_0,\ s_0{:}a,\ s_0;z_1].$$

In other words, we prove, for any input object a, the existence of a plan z_1 such that, for any initial state s_0, if we are in s_0 and execute plan z_1, we obtain a state $s_0;z_1$ in which the goal condition $\mathcal{Q}$ is true.
For example, the plan to construct a building b, that is, to achieve the condition

$$:constructed(b,\ s_f)$$

is derived by proving the theorem

$$(\forall b)(\exists z_1)(\forall s_0){:}constructed(s_0{:}b,\ s_0;z_1).$$

In skolemizing this formula, we obtain the sentence

$$:constructed(s_0{:}b,\ s_0;z_1)$$

where s_0 and b are skolem functions and constants respectively and z_1 is a variable. Because b is a rigid designator, this is equivalent to the sentence

$$:constructed(b,\ s_0;z_1).$$

To prove this theorem, we establish the goal

Assertion	Goal	$s_0;construct(b)$
	$:constructed(b,\ s_0;z_1)$	$s_0;z_1$

that specifies the condition that b is constructed by the execution of the plan z_1 from the initial state s_0. Henceforth, we will refer to such a condition as the *goal condition*. In our notation [MW90], each row contains a sentence, either an *assertion* or a *goal*, and a term, the *plan entry*, for each output of the desired plan. In the example above, the sentence is the goal *:constructed*$(b,\ s_0;z_1)$ and the plan entry for the desired plan s_0;*construct*(b) is the term $s_0;z_1$. The deduction process proceeds by the application of sound deduction rules, which add new rows to the initial one. As a side effect, variable z_1 in the plan entry is instantiated to different terms. The structure of such terms, and how it is determined by the deduction process, is the subject of Sections 4–8. The proof terminates when the propositional constant *true* is derived as a goal, or the propositional constant *false* is derived as an assertion. The plan extracted from the proof is the plan entry associated with the final row.

To simplify the presentation, in the following sections we will only apply the resolution rule to derive combined plans. We illustrate the application of the resolution rule in plan theory with a simple example from the blocks world.

Example (resolution rule in plan theory). Consider the axiom

if :clear$(x,\ s)$ *then*	*:on*$(x,\ table,\ ;put(x,\ table,\ s))$ $^-$		

asserting that after a block x has been put on the table, the block will indeed be on the table, provided that it was clear beforehand. The goal that we want to establish is

	:on$(a,\ table,\ s_0;z_1)$ $^+$		$s_0;z_1$

In other words we look for a plan z_1 such that after its execution the block a is on the table. The positive and negative signs of boxed subsentences indicate their *polarity*. In particular, negative polarity means that the subsentence is within the scope of an odd number of implicit or explicit negations, while positive polarity indicates that the subsentence is within the scope of an even number of implicit or explicit negations. Non-negated goals have positive polarity, while non-negated assertions have negative polarity, because we could push them into the goal column by negating them. For the resolution rule to be applicable, two subsentences must be unifiable and of opposite polarity. The boxed subsentences are unifiable, with a most general unifier

$$\{x \leftarrow a,\ s \leftarrow s_0,\ z_1 \leftarrow put(a,\ table)\}.$$

The equivalence of the boxed subsentences is obtained by the equatioanal-unification algorithm [Fay79], that invokes certain of the equations and equivalences of a theory, such as *linkage* and *rigidity* axioms, so they need not be included among the assertions. In the above case, the equational-unification algorithm invokes the property

$$s_0;put(a,\ table)\ =\ ;put(s_0;a,\ s_0;a:table,\ s_0;a;table)$$

which is an instance of the *plan linkage* axiom schema, and the rigidity and applicability of the constants a and *table*. In this case, the resolution rule consists of applying the most general unifier to the two sentences, replacing the negative-polarity subsentence with *false* and the positive-polarity subsentence with *true*, and introducing a new row in which the conjunction of the resulting sentences is stated as a goal. Applying the resolution rule and simplifying, we obtain the new row

	$:clear(a,\ s_0)$	$s_0;put(a,\ table)$

meaning that the specified goal to have block a on the table is achieved by executing the plan $put(a,\ table)$ in state s_0, provided that a is clear in s_0. □

3 A Construction Theory

The deductive techniques outlined in the previous section are applied to construction planning, a fundamental activity in the management and execution of construction projects.

The construction planning problem can be defined as the selection of a course of actions whose execution results in the construction of a building. Consider, for instance, the single-story building of Fig. 1, composed of two columns and a beam. A possible plan to construct this building is

$$constr(story) \Leftarrow \left\{ \begin{array}{l} \left(\begin{array}{l} place(c_1) \parallel \\ place(c_2) \end{array} \right); \\ place(beam) \end{array} \right.$$

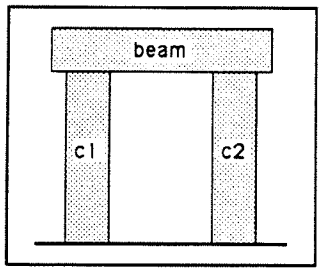

Fig. 1. A single-story building.

meaning that to construct the single-story building *story*, first column c_1 and column c_2 are placed in parallel and then *beam* is placed. The function "||" denotes parallel composition while ";" denotes sequential composition.

A simple *construction theory* to express facts about the construction domain can be developed. Type predicates are omitted for brevity. Instead, a convention on variable names is introduced: x, y range over buildings, u, v over stories, s over states and z, z_1, z_2 over plans. For example, the fact that the *place* action has two preconditions is expressed

by the axiom

if :placed(support(x), s) and not :placed(x, s)
then :placed(x, ;place(x, s))

meaning that for x to be *placed* in state s, the support of x must be placed in s, but x must not. The function *support*, applied to an object x, returns the complete set of objects that are needed by x as a support. For instance, the support of *beam* in Fig. 1 is the set containing columns c_1 and c_2. The following conventions are introduced on constants: s_0 denotes the intial state, b the input building, ε the empty building and Λ the empty plan.

We can think of a building x as a composite structure. The function *lower*(x) returns the first story of x, and the function *upper*(x) returns the rest of x, i.e., the subbuilding made of all the stories except the first one. Every nonempty building is then composed of a lower and an upper part. This is expressed by the *building decomposition* axiom

if not $(x = \varepsilon)$
then $x = (lower(x) \diamond upper(x))$

where "ε" is the empty building and "$\diamond$" is a function that composes a given story and part of a building to obtain a new building such that the given story is the first floor of the new building and the given part is the rest of the new building. Next, we assume that actions are not destructive, with the *frame* axiom

if :constructed(u, s)
then :constructed(u, s;z)

meaning that if u is constructed in state s, then it is also constructed after the execution in state s of any plan z. Finally, the *upper* axiom

if not $(x = \varepsilon)$
then $upper(x) \prec_{build} x$

states that the upper part of a nonempty building is properly included in the building itself, where $\prec_{build}$ denotes proper building inclusion.

These are examples of axioms for a possible construction theory. Other axioms are given in the derivation steps presented in the following sections and in the Appendix, which shows the derivation of a plan to construct the multi-story building of Fig. 2. The plan extracted from that proof is

$$construct(b) \Leftarrow \begin{cases} if\ (b = \varepsilon) \\ then\ \Lambda \\ else\ constr(lower(b)); \\ \quad construct(upper(b)) \end{cases}$$

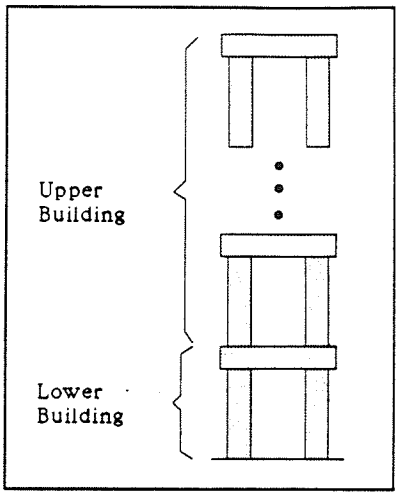

Fig. 2. A multi-story building.

meaning that if b is the empty building, the empty plan Λ will work; otherwise, first construct the story given by the lower part of b and then construct the upper part of b. We are now ready to demonstrate how the derivation process determines the structure of synthesized plans in the domain of construction planning. Some familiarity with the deductive techniques described in [MW87b, MW90] is assumed.

4 Sequencing

We can combine two plans in several ways. For instance, one plan could be executed first and the other one next. In such a case, we talk of *sequential composition* of plans and represent it with the operator "; ". Given two plan terms p_1 and p_2, the plan term $p_1;p_2$ denotes the sequential execution of p_1 and p_2. Executing plan $p_1;p_2$ is the same as executing first plan p_1 and then plan p_2. This is expressed by the *sequential composition* axiom

$$s;(p_1;p_2) = (s;p_1);p_2$$

for all states s and plans p_1 and p_2. Note that we use the semicolon to stand for both the state produced by evaluating a plan term and the composition of two plan terms. Due to the previous axiom, this should not generate confusion.

To illustrate the introduction of sequential composition in a derived plan, consider the axiom (A1 in the *constr*(*story*) plan)

if :*placed*(*support*(*x*), *s*) *and* *not* :*placed*(*x*, *s*) *then* [:*placed*(*x*, ;*place*(*x*, *s*))]$^-$		

asserting that x is *placed* after the *place* action is executed in state s, provided that it was not *placed* in s but its *support* was. Also, assume that we have derived the goal

	[*placed*(*c*, $s_0;z_1$)]$^+$		$s_0;z_1;$*place*(*beam*)

meaning that if, after the execution of plan z_1 column c is *placed*, then placing *beam* after the execution of z_1 achieves the goal condition. The boxed subsentences are unifiable, with a most general unifier

$$\{x \leftarrow c,\ s \leftarrow s_0;z_2,\ z_1 \leftarrow z_2;place(c)\}.$$

The equational-unification algorithm invokes the *place linkage* axiom

$$s_0;z_2;place(c) = ;place(s_0;z_2{:}c,\ s_0;z_2;c)$$

and the rigidity and applicativity of the constant c. Applying the resolution rule, we obtain

	:*placed*(*support*(*c*), $s_0;z_2$) *and* *not* :*placed*(*c*, $s_0;z_2$)	$s_0;z_2;$*place*(c_2); *place*(*beam*)

In other words, the goal condition is achieved by the execution of the plan step z_2 followed by placing c and *beam*, provided that the *support* of c is *placed* after the execution of z_2, whereas c is not. The three plan steps z_2, *place*(c) and *place*(*beam*) are sequentially composed in the plan entry of the new goal. They will be so combined also in the final plan, provided that we establish this goal.

5 Parallelism

As we introduced the operator "; " for sequential composition of plans, we now introduce the operator "||" for *parallel composition*. Given two plan terms p_1 and p_2, their *parallel composition* $p_1 \parallel p_2$ is also a plan term, denoting the plan in which p_1 and p_2 are executed in parallel. If s is a state term, $s;(p_1 \parallel p_2)$ denotes the state obtained by the parallel execution of p_1 and p_2 in state s.
Parallelism can be introduced in a plan through the use of parallelism axioms such as (A2 in the *constr*(*story*) proof)

if (*:placed*(*support*(*x*), *s*) *and* *:placed*(*support*(*y*), *s*) *and* *not :placed*(*x*, *s*) *and not :placed*(*y*, *s*) *and :available*(*teams*, *2*, *s*)) *then* *:placed*(*s:x*, *s;*(*place*(*x*)||*place*(*y*))) *and* *:placed*(*s:y*, *s;*(*place*(*x*)||*place*(*y*))) −		

asserting that if the support of the objects x and y has been placed, while the two objects have not been placed yet and two construction teams are available, then both x and y will be placed as a result of placing them in parallel. Now consider the goal (G30 in the *constr*(*story*) proof)

	not :placed(*beam*, $s_0;z_2$) *and* *:placed*(c_1, $s_0;z_2$) *and* *:placed*(c_2, $s_0;z_2$) +	$s_0;z_2;$*place*(*beam*)

meaning that if, after the execution of some plan step z_2 the *beam* is not placed but c_1 and c_2 are, then we can achieve our goal condition by first executing plan z_2 and then placing the *beam*. The boxed subsentences are unifiable, with a most general unifier

$$\{x \leftarrow c_1,\ y \leftarrow c_2,\ s \leftarrow s_0,\ z_2 \leftarrow place(c_1)||place(c_2)\}.$$

The equational-unification algorithm invokes the rigidity and applicativity of c_1 and c_2. Applying the resolution rule, we obtain (G31 in the *constr*(*story*) proof)

	not :placed(*beam*, $s_0;$(*place*(c_1)||*place*(c_2))) *and :placed*(*support*(c_1), s_0) *and :placed*(*support*(c_2), s_0) *and not :placed*(c_1, s_0) *and not :placed*(c_2, s_0) *and :available*(*teams*, *2*, s_0)	$s_0;$ (*place*(c_1)||*place*(c_2)); *place*(*beam*)

In other words, if after the execution of the plan *place*(c_1)||*place*(c_2) the *beam* is not placed, and if in the initial state s_0 the two columns c_1 and c_2 are not placed but their supports are and two construction teams are available, then, to achieve the goal condition, we can execute the plan *place*(c_1)||*place*(c_2) followed by the plan *place*(*beam*). As we see, the parallel composition operator "||" appears now in the plan entry of the new goal. This will lead to the final plan for the building in Fig. 1 presented in Sect. 3.

6 Contingency

The nonclausal resolution principle presented for program synthesis in [MW85] accounts for the introduction of conditionals whenever it is applied between two rows both having plan entries. Consider two goal rows (similarly for two assertions or for an assertion and a goal) in which a matching proposition P occurs in state S with opposite polarity and having S as a common initial segment in the plan entries. When the resolution rule is applied as explained in Section 2, the plan entry of the new goal has S as its initial segment; its final segment is a conditional such that the test is the matched proposition P, the *then*-clause is the final segment of the row in which P occurs positively and the *else*-clause is the final segment of the row in which P occurs negatively.

The rationale behind the rule is a case analysis on the truth of P. If P is true, the goal in which P occurs positively is also true, and hence the associated plan entry satisfies the specified condition. Similarly, if P is false, the goal in which P occurs negatively is true, and hence the associated plan entry satisfies the goal condition.

Consider, for instance, the problem of generating a plan for placing the two columns c_1 and c_2 of the single-story building in Fig. 1, in the case in which the human-resource availabilty is not known at planning time, i.e., we do not know whether one or two construction teams will be available for the task. The proof is carried out along two branches, one for each hypothesis. In one case, the goal

	$:available(teams,\ 2,\ s_0)$	$+$		$s_0;(place(c_1)\|\|place(c_2))$

is derived, meaning that if two construction teams are available in the initial state s_0, the goal condition is achieved by placing c_1 and c_2 in parallel. In the other case, the goal

	not	$:available(teams,\ 2,\ s_0)$	$-$	$s_0;place(c_1);place(c_2)$

is obtained, stating that the goal condition is achieved by placing first column c_1 and then column c_2, provided that two construction teams are not available.

The common initial segment S of the two output entries is s_0 and the matching proposition P occurring in s_0 with opposite polarity is $:available(teams,\ 2,\ s_0)$. Applying the resolution rule with empty most general unifier, we obtain the final goal

	$true$	$s_0;$ $\left(\begin{array}{l} if\ :available(teams,\ 2) \\ then\ place(c_1)\|\|place(c_2) \\ else\ place(c_1);place(c_2) \end{array} \right)$

where a conditional is introduced into the plan entry, which is also the final plan. Its meaning is that the two columns c_1 and c_2 can be placed in parallel if two construction teams are available, but they have to be placed sequentially if there is only one team.

The proof to derive the *constr(story)* plan in the Appendix presents a different case in which the information on the team availability is known at planning time. The axiom (A11 in the *constr(story)* proof)

:available(teams, 2, s)		

states that two construction teams are available in every state s of the construction process. This information is exploited in the proof (resolution rule between A11 and G33) to introduce the parallel composition $place(c_1)||place(c_2)$ in the final plan without need to conduct a case analysis on the resource availability.

Another example of contingency is introduced in the recursive plan *construct(b)* to distinguish the base case from the induction step.

7 Repetition

As soon as a task reaches a considerable degree of complexity, it is likely to contain repetitions, i.e., sequences of actions that must be performed several times. Consider the proof to derive a plan to construct the building b in Fig. 2. The initial goal is (G1 in the *construct(b)* proof)

	:constructed(b, s_0;z_1)	s_0;z_1

By application of the induction rule [MW87b] we obtain the new row (A6 in the *construct(b)* proof)

if $<s{:}x,\ s> \prec_\alpha <b,\ s_0>$ *then* *:constructed(x, s;construct(x))*		

The induction rule, beyond the scope of this paper, introduces automatically the induction hypothesis on the plan being derived. In other words, it assumes inductively that the plan *construct(x)* that we are computing satisfies the goal condition *constructed* for any input x in any state s, provided that the pair $<s{:}x,\ s>$ is less then the pair $<b,\ s_0>$ with respect to some well-founded relation $\prec_\alpha$, not yet determined (a well-founded relation is one that allows no infinite decreasing sequences; the less-than relation $<$ over the nonnegative integers is a typical example).

Assume that in the derivation we have obtained the goal (G5 in the *construct(b)* proof)

	not $(b = \varepsilon)$ *and* *:constructed*$(lower(b),\ s_0;z_1)$ *and* $\boxed{:constructed(upper(b),\ s_0;z_1)}^{+}$	$s_0;z_1$

meaning that if b is not the empty building and if after the execution of plan z_1 both the lower and upper parts of b are constructed, then the execution of z_1 achieves the goal condition. The boxed subsentences of the last two rows are unifiable, with a most general unifier

$$\{x \leftarrow upper(b),\ s \leftarrow s_0;z_2,\ z_1 \leftarrow z_2;construct(upper(b))\}.$$

The equational-unification algorithm invokes the *construct linkage* axiom

$$s_0;z_2;construct(upper(b)) = ;construct(s_0;z_2;upper(b),\ s_0;z_2)$$

and the rigidity of $upper(b)$. Applying the resolution rule we obtain the goal (G7 in the $construct(b)$ proof)

	$<s_0;z_2:upper(b),\ s_0;z_2> \prec_\alpha <b,\ s_0>$ *and not* $(b = \varepsilon)$ *and :constructed*$(lower(b),$ $s_0;z_2;construct(upper(b)))$	$s_0;z_2;construct(upper(b))$

that introduces in the plan entry a recursive call $construct(upper(b))$ of the plan *construct* that we are computing on the part $upper(b)$ of b. The condition $<s_0;z_2:upper(b),\ s_0;z_2> \prec_\alpha <b,\ s_0>$ and the fact that $\prec_\alpha$ is well-founded guarantee that the recursive call will not lead to a nonterminating computation. The relation $\prec_\alpha$, will be instantiated to the *proper subbuilding* relation $\prec_{build}$ described in Section 3.

Repetition can also be obtained through iteration. For a discussion on how to generate an iterative plan from the transformation of a recursive one see [BD77].

8 Modularity

It is a well-established principle of software engineering that the modular design of programs improves modifiability, understandability and reliability. The same principle applies to plans [Sac73]. Furthermore, the search space can be significantly reduced by first planning at abstract level and then expanding the abstract plans into more detailed plans. Once we have derived a plan, we can use it as a subplan in future derivations. We do this by including an assertion stating that the derived plan does indeed meet its specification. Suppose we have derived a plan *constr*, to constuct a single-story building u that meets the specification $:constructed(u,\ s_0;constr(u))$. Then, in the derivation of a new plan *construct*

to construct a multi-story building, we may include the assertion (A10 in the *construct*(b) proof)

	:constructed(u, s;*constr*(u)) $^-$		

stating that *constr* does satisfy its specification. Such an assertion can then be used in the proof. Assume we have derived the goal (G9 in the *construct*(b) proof)

	$<s_0;z_2$:*upper*(b), $s_0;z_2> \prec_\alpha <b, s_0>$ *and not* ($b = \varepsilon$) *and* *:constructed*(*lower*(b), $s_0;z_2$)$^+$	$s_0;z_2$;*construct*(*upper*(b))

meaning that to construct b we must first execute some plan step z_2 and then construct the uppper part of b, provided that the pair $<s_0;z_2$:*upper*(b), $s_0;z_2>$ is less than the pair $<b$, $s_0>$ with respect to some well-founded relation $\prec_\alpha$, that b is not the empty building and that the lower part of b is constructed after the execution of z_2. The boxed subsentences of these two rows are unifiable, with a most general unifier

$$\{u \leftarrow \mathit{lower}(b),\ s \leftarrow s_0,\ z_2 \leftarrow \mathit{constr}(\mathit{lower}(b))\}.$$

Applying the resolution rule, we obtain (G11 in the *construct*(b) proof)

	$<s_0;z_2$:*upper*(b), $s_0;z_2> \prec_\alpha <b, s_0>$ *and not* ($b = \varepsilon$)	s_0;*constr*(*lower*(b)); *construct*(*upper*(b))

In other words, to construct b, first construct the story *lower*(b) and then construct the building *uppper*(b), provided that the pair $<s_0;z_2$:*upper*(b), $s_0;z_2>$ is less then the pair $<b$, $s_0>$ with respect to some well-founded relation $\prec_\alpha$, and that b is not the empty building. As we see, the subplan *constr* is introduced in the plan entry, and in the final plan, provided that we can establish this goal.

9 Discussion

The deductive steps that introduce relevant features in construction plans have been illustrated with examples taken from a proof for a multi-story building. The logical framework, an adaptation of situational calculus for automated planning, is domain-independent and not committed to any specific vocabulary. In the field of civil engineering, it allows us to synthesize construction plans containing desirable features such as sequencing, parallelism, contingency, repetition and modularity. In particular, such powerful constructs as contingency and repetition enable the system to account for conditions not known at planning

time and to return a compact representation of repetitive patterns. Such an expressive power is not achieved by other construction planning systems.

The plan illustrated in this paper was derived with the Deductive Tableau System [BMW90]. Originally, the system was an interactive implementation of the theorem-proving framework described in [MW90, MW85]. Then, it was extended to allow the synthesis of programs. Presently, we are testing a new version of the system that generates plans either automatically or interactively. In the latter case, the user can control the derivation process to introduce desired features into the plan.

Fragmentation between different phases of a project, such as design and planning, has a relevant impact on the construction industry [HLP+89]. For this reason, considerable research in civil engineering is now concentrating on the automatic generation of symbolic description from architectural drawings. Encouraging results [IULD89, CLS91] indicate that it should be possible to integrate Deductive Tableau with a 3-D CAD system to automatically extract the axioms describing the structure of the facility to be constructed from purely geometrical information.

Appendix

Plan: *construct*(b)

No	Asrt	Goal	$s_0;construct(b)$	Explanation
G1		$:constructed(\boxed{b}, s_0;z_1)^+$	$s_0;z_1$	initial goal
A2	*if not* $(x = \varepsilon)$ *then* $\boxed{x = lower(x) \diamond upper(x)}^-$			building decomposition axiom
G3		*not* $(b = \varepsilon)$ *and* $\boxed{:constructed(lower(b) \diamond upper(b), s_0;z_1)}$	$s_0;z_1$	eq G1,A2
A4	$\boxed{:constructed(u \diamond x, s) \equiv :constructed(u, s) \text{ and } :constructed(x, s)}^-$			constructed distribution axiom
G5		*not* $(b = \varepsilon)$ *and* $:constructed(lower(b), s_0;z_1)$ *and* $\boxed{:constructed(upper(b), s_0;z_1)}^+$	$s_0;z_1$	eqv G3,A4
A6	*if* $<s{:}x, s> \prec_\alpha <b, s_0>$ *then* $\boxed{:constructed(x, s;construct(x))}^-$			induction G1
G7		$<s_0;z_2{:}upper(b), s_0;z_2> \prec_\alpha <b, s_0>$ *and not* $(b = \varepsilon)$ *and* $\boxed{:constructed(lower(b), s_0;z_2;construct(upper(b)))}^+$	$s_0;z_2;construct(upper(b))$	res G5,A6
A8	*if* $:constructed(u, s)$ *then* $\boxed{:constructed(u, s;z)}^-$			frame axiom
G9		$<s_0;z_2{:}upper(b), s_0;z_2> \prec_\alpha <b, s_0>$ *and not* $(b = \varepsilon)$ *and* $\boxed{:constructed(lower(b), s_0;z_2)}^+$	$s_0;z_2;construct(upper(b))$	res G7,A8
A10	$\boxed{:constructed(u, s;constr(u))}^-$			*constr* specification axiom
G11		$\boxed{<s_0;z_2{:}upper(b), s_0;z_2> \prec_\alpha <b, s_0>}$ *and not* $(b = \varepsilon)$	$s_0;constr(lower(b));$ $construct(upper(b))$	res G9,A10

A12	$\boxed{<x_1,\ s_1> \prec_{\Pi_1(\beta)} <x_2,\ s_2> \equiv x_1 \prec_\beta x_2}^-$			second-projection axiom
G13		$\boxed{upper(b) \prec_\alpha b}^+$ *and not* $(b = \varepsilon)$	s_0;*constr*(*lower*(b)); *construct*(*upper*(b))	eqv G11,A12
A14	*if not* $(x = \varepsilon)$ *then* $\boxed{upper(x) \prec_{build} x}^-$			upper axiom
G15		*not* $\boxed{(b = \varepsilon)}^-$	s_0;*constr*(*lower*(b)); *construct*(*upper*(b))	res G13,A14
G16		$\boxed{:constructed(\varepsilon,\ s_0;z_1)}^+$	s_0; $\left(\begin{array}{l} \textit{if } (b = \varepsilon) \\ \textit{then } z_1 \\ \textit{else constr}(\textit{lower}(b)); \\ \quad \textit{construct}(\textit{upper}(b)) \end{array}\right)$	eq G1,G15
A17	$\boxed{:constructed(\varepsilon,\ s;\Lambda)}^-$			empty building axiom
G18		*true*	s_0; $\left(\begin{array}{l} \textit{if } (b = \varepsilon) \\ \textit{then } \Lambda \\ \textit{else constr}(\textit{lower}(b)); \\ \quad \textit{construct}(\textit{upper}(b)) \end{array}\right)$	res G16,A17

The final plan is:

$$\textit{construct}(b) \Leftarrow \left\{\begin{array}{l} \textit{if } (b = \varepsilon) \\ \textit{then } \Lambda \\ \textit{else constr}(\textit{lower}(b)); \\ \quad \textit{construct}(\textit{upper}(b)) \end{array}\right.$$

Plan: *constr(story)*

No	Assertion	Goal	s_0;*constr(story)*	Explanation
A1	*if :placed(support(x), s) and not :placed(x, s) then* [*:placed(x, ;place(x, s))*]$^-$			sequencing
A2	*if (:placed(support(x), s) and :placed(support(y), s) and not :placed(x, s) and not :placed(y, s) and :available(teams, 2, s)) then* [*(:placed(s:x, s;(place(x)||place(y))) and :placed(s:y, s;(place(x)||place(y))))*]$^-$			parallelism
A3	*if (not x=y) then* [*:placed(x, s) ≡ :placed(x, ;place(y, s))*]$^-$			sequencing frame axiom
A4	*if (not x=y and not x=z) then* [*:placed(x, s) ≡ :placed(x, ;place(y, s)||place(z, s))*]$^-$			parallel frame axiom
A5	[*:placed([x|l], s) ≡ (:placed(x, s) and :placed(l, s))*]$^-$			list distribution
A6	[*:placed([], s)*]$^-$			empty list
A7	[*support(c_1)=[ground]*]$^-$			column 1
A8	[*support(c_2)=[ground]*]$^-$			column 2
A9	[*support(beam)=[c_1,c_2]*]$^-$			beam
A10	[*:placed(ground, s)*]$^-$			ground
A11	*not* [*:available(teams, 2, s)*]$^+$			initial state
A12	*not* [*:placed(c_1, s_0)*]$^+$			initial state
A13	*not* [*:placed(c_2, s_0)*]$^+$			initial state
A14	*not* [*:placed(beam, s_0)*]$^+$			initial state
A15	[*story = [c_1,c_2,beam]*]$^-$			story definition

G16		$:placed(\boxed{story}, s_0;z_1)$	$s_0;z_1$	initial goal
G17		$\boxed{:placed([c_1,c_2,beam], s_0;z_1)}$	$s_0;z_1$	eq A15,G16
G18		$:placed(c_1, s_0;z_1)$ *and* $\boxed{:placed([c_2,beam], s_0;z_1)}$	$s_0;z_1$	eqv A5,G17
G19		$:placed(c_1, s_0;z_1)$ *and* $:placed(c_2, s_0;z_1)$ *and* $\boxed{:placed([beam], s_0;z_1)}$	$s_0;z_1$	eqv A5,G18
G20		$:placed(c_1, s_0;z_1)$ *and* $:placed(c_2, s_0;z_1)$ *and* $:placed(beam, s_0;z_1)$ *and* $\boxed{:placed([], s_0;z_1)}^+$	$s_0;z_1$	eqv A5,G19
G21		$:placed(c_1, s_0;z_1)$ *and* $:placed(c_2, s_0;z_1)$ *and* $\boxed{:placed(beam, s_0;z_1)}^+$	$s_0;z_1$	res A6,G20
G22		$:placed(support(beam), s_0;z_2)$ *and* *not* $:placed(beam, s_0;z_2)$ *and* $\boxed{:placed(c_1, s_0;z_2;place(beam))}$ *and* $:placed(c_2, s_0;z_2;place(beam))$	$s_0;z_2;place(beam)$	res A1,G21
G23		$:placed(support(beam), s_0;z_2)$ *and* *not* $:placed(beam, s_0;z_2)$ *and* *not* $\boxed{c_1=beam}^-$ *and* $:placed(c_1, s_0;z_2)$ *and* $:placed(c_2, s_0;z_2;place(beam))$	$s_0;z_2;place(beam)$	eqv A3,G22
G24		$:placed(support(beam), s_0;z_2)$ *and* *not* $:placed(beam, s_0;z_2)$ *and* $:placed(c_1, s_0;z_2)$ *and* $\boxed{:placed(c_2, s_0;z_2;place(beam))}$	$s_0;z_2;place(beam)$	res not (c_1=beam), G23
G25		$:placed(support(beam), s_0;z_2)$ *and* *not* $:placed(beam, s_0;z_2)$ *and* *not* $\boxed{c_2=beam}^-$ *and* $:placed(c_1, s_0;z_2)$ *and* $:placed(c_2, s_0;z_2)$	$s_0;z_2;place(beam)$	eqv A3,G24
G26		$:placed(\boxed{support(beam)}, s_0;z_2)$ *and* *not* $:placed(beam, s_0;z_2)$ *and* $:placed(c_1, s_0;z_2)$ *and* $:placed(c_2, s_0;z_2)$	$s_0;z_2;place(beam)$	res not (c_2=beam), G25
G27		$\boxed{:placed([c_1,c_2], s_0;z_2)}$ *and* *not* $:placed(beam, s_0;z_2)$ *and* $:placed(c_1, s_0;z_2)$ *and* $:placed(c_2, s_0;z_2)$	$s_0;z_2;place(beam)$	eq A9,G26

G28		$\boxed{:placed([c_2],\ s_0;z_2)}$ *and* *not* $:placed(beam,\ s_0;z_2)$ *and* $:placed(c_1,\ s_0;z_2)$ *and* $:placed(c_2,\ s_0;z_2)$	$s_0;z_2;place(beam)$	eqv A5,G27
G29		$\boxed{:placed([],\ s_0;z_2)}^{+}$ *and* *not* $:placed(beam,\ s_0;z_2)$ *and* $:placed(c_1,\ s_0;z_2)$ *and* $:placed(c_2,\ s_0;z_2)$	$s_0;z_2;place(beam)$	eqv A5,G28
G30		*not* $:placed(beam,\ s_0;z_2)$ *and* $\boxed{:placed(c_1,\ s_0;z_2)\ and\ :placed(c_2,\ s_0;z_2)}^{+}$	$s_0;z_2;place(beam)$	res A6,G29
G31		*not* $:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ *and* $:placed(support(c_1),\ s_0)$ *and* $:placed(support(c_2),\ s_0)$ *and* *not* $\boxed{:placed(c_1,\ s_0)}^{-}$ *and not* $:placed(c_2,\ s_0)$ *and* $:available(teams,\ 2,\ s_0)$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res A2,G30
G32		*not* $:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ *and* $:placed(support(c_1),\ s_0)$ *and* $:placed(support(c_2),\ s_0)$ *and not* $\boxed{:placed(c_2,\ s_0)}^{-}$ *and* $:available(teams,\ 2,\ s_0)$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res A12,G31
G33		*not* $:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ *and* $:placed(support(c_1),\ s_0)$ *and* $:placed(support(c_2),\ s_0)$ *and* $\boxed{:available(teams,\ 2,\ s_0)}^{-}$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res A13,G32
G34		*not* $:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ *and* $\boxed{:placed(support(c_1),\ s_0)}$ *and* $:placed(support(c_2),\ s_0)$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res A11,G33
G35		*not* $:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ *and* $:placed([ground],\ s_0)$ *and* $\boxed{:placed(support(c_2),\ s_0)}$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	eq A7,G34
G36		*not* $:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ *and* $\boxed{:placed([ground],\ s_0)}$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	eq A8,G35
G37		*not* $:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ *and* $\boxed{:placed(ground,\ s_0)}^{+}$ *and* $:placed([],\ s_0)$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	eqv A5,G36

G38		$not\ :placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))$ $and\ \boxed{:placed([],\ s_0)}^{+}$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res A10,G37
G39		$not\ \boxed{:placed(beam,\ s_0;(place(c_1)\|\|place(c_2)))}$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res A6,G38
G40		$not\ \boxed{beam{=}c_1}^{-}\ and\ not\ beam{=}c_2\ and$ $not\ :placed(beam,\ s_0)$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	eqv A4,G39
G41		$not\ \boxed{beam{=}c_2}^{-}\ and\ not\ :placed(beam,\ s_0)$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res not (beam=c_1), G40
G42		$not\ \boxed{:placed(beam,\ s_0)}^{-}$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res not (beam=c_2), G41
G43		$true$	$s_0;$ $(place(c_1)\|\|place(c_2));$ $place(beam)$	res A14,G42

The final plan is:

$$constr(story) \Leftarrow \left\{ \begin{array}{l} \left(\begin{array}{l} place(c_1)\ \| \\ place(c_2) \end{array} \right); \\ place(beam) \end{array} \right.$$

References

[BD77] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24:44–67, 1977.

[BMW90] R. Burback, Z. Manna, and R. Waldinger. *Using the Deductive Tableau System.* Chariot Software Group, 1990.

[CLS91] J. Cherneff, R. Logcher, and D Sriram. Integrating CAD with construction-schedule generation. *Journal of Computing in Civil Engineering*, 5, 1991.

[DLHR89] A. Darwiche, R. Levitt, and B. Hayes-Roth. Oarplan: Generating project plans in a blackboard system by reasoning about objects, actions and resources. *Journal of Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, 3, 1989.

[Fay79] M. Fay. First-order unification in an equational theory. *Proceedings of the Fourth Workshop on Automated Deduction*, pages 161–167, 1979.

[FN71] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Geo87] M. Georgeff. Planning. *Annual Review of Computer Science*, 2, 1987.

[Gre69] C. C. Green. Application of theorem proving to problem solving. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 219–239, 1969.

[HLP+89] H. C. Howard, R. E. Levitt, B. C. Paulson, J. G. Pohl, and C. B. Tatum. Computer integration: Reducing fragmentation in AEC industry. *Journal of Computing in Civil Engineering*, 3, 1989.

[HZGR+87] C. Hendrickson, C. Zozaya-Gorostiza, D. Rehak, E. Baracco-Miller, and P. Lim. Expert systems for construction planning. *Journal of Computing in Civil Engineering*, 1, 1987.

[IULD89] K. Ito, Y. Ueno, R. Levitt, and A. Darwiche. Linking knowledge-based systems to CAD design data with an object-oriented model. Technical Report 17, Center for Integrated Facility Engineering, Stanford University, 1989.

[Kar89] N. Kartam. *Investigating the Utility of Artificial Intelligence Techniques for the Automatic Generation of Construction Projects Plans.* PhD thesis, Dept. of Civil Engineering, Stanford University, 1989.

[KLW91] N. A. Kartam, R. E. Levitt, and D. E. Wilkins. Extending artificial intelligence techniques for hierarchical planning. *Journal of Computing in Civil Engineering*, 5, 1991.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4, 1969.

[MW85] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming*, volume 1. Addison Wesley, 1985.

[MW87a] Z. Manna and R. Waldinger. The deductive synthesis of imperative LISP programs. *Sixth AAAI National Conference on Artificial Intelligence*, 1987.

[MW87b] Z. Manna and R. Waldinger. How to clear a block: a theory of plans. *Journal of Automated Reasoning*, 1987.

[MW90] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming*, volume 2. Addison Wesley, 1990.

[NSL88] D. Navinchandra, D. Sriram, and R. Logcher. Ghost: A project network generator. *Journal of Computing in Civil Engineering*, 2, 1988.

[Ros81] S. Rosenschein. Plan synthesis: a logical perspective. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 331–337, 1981.

[Sac73] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 115–135, 1973.

[THD90] A. Tate, J. Hendler, and M. Drummond. A review of ai planning techniques. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 36–53. Morgan Kaufmann, 1990.

[Wal81] R. Waldinger. Achieving several goals simultaneously. In N. Nilsson and B. Webber, editors, *Readings in Artificial Intelligence*, pages 250–271. Tioga, Palo Alto, 1981.